Debugging iOS Applications With IDA

IDA 6.95 introduced the Remote iOS Debugger plugin, which allows users to debug iOS target applications directly from IDA. It works on all supported platforms (Mac, Windows, Linux), supports both ARM32 and ARM64 targets, and has been extensively tested with iOS 9.

The goal of this tutorial is to install a small example app on your device and use IDA to debug it.

Before we begin, please note that the Remote iOS Debugger requires a running instance of the Apple iOS debugserver in order to function. Since iOS devices are often jailed, spawning a remote debugger process (or doing anything else for that matter) can be somewhat of a task.

IDA provides various ways of working with jailed devices, depending on your platform. If you are a Mac user, continue reading. If you are a non-mac user, skip to 'Non-Mac users'.

Quick Start

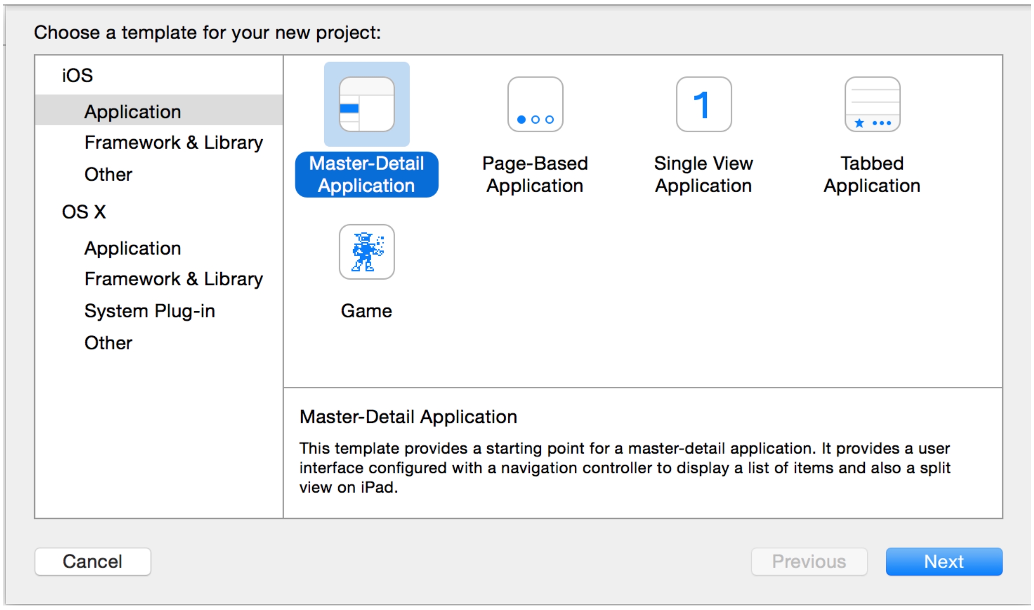
The fastest way to get started is to use Xcode to build and install a sample app, then switch to IDA to debug it. We will cover this process in depth here. If you prefer not to use Xcode, jump ahead to the 'ios_deploy' section.

Creating a Sample Project

Note that since IDA depends on the iOS debugserver, IDA can only debug applications that the debugserver can debug. Usually, the default debugserver installed by Xcode can only debug applications that you have built with Xcode.

Thus, we must start by building a sample app ourselves, just for this tutorial.

First open Xcode.app and on the 'Welcome to Xcode' screen, choose 'Create a new Xcode project'. Then when asked to choose a template, select one of the iOS Application templates:



Choose a template for your new project:

Master-Detail Application

This template provides a starting point for a master-detail application. It provides a user interface configured with a navigation controller to display a list of items and also a split view on iPad.

Click Next, and when asked for the project options, use the following values:

Product Name: demo

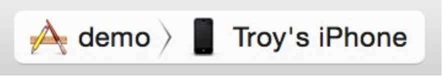
Organization Identifier: me

Language: Objective-C

For the remaining fields you can use the default values.

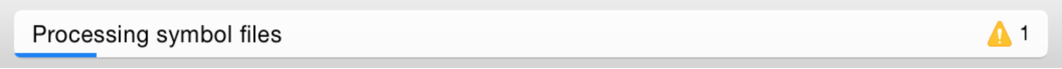
Symbols

Once your project has been created, ensure that your device is attached to your host machine and is selected as the current device in the top left of the Xcode window:



When you first select your device, Xcode will perform two important tasks. First it will install the debugserver on your device (which IDA will need for debugging), and it will also extract symbol files from dyld_shared_cache.

Symbol processing usually takes a while, and you can check the progress at the top of the Xcode window:



When this process is completed, Xcode will store the symbols in:

```
~/Library/Developer/Xcode/iOS DeviceSupport/<iOS version>/Symbols
```

Ensure that this directory is present and use it to set `SYMBOL_PATH` in `idaq.app/Contents/MacOS/cfg/dbg_ios.cfg`:

```
SYMBOL_PATH = "~/Library/Developer/Xcode/iOS DeviceSupport/9.3.1 (13E238)/Symbols";
```

Installation

In order for IDA to debug this app, it must know the path to the app's executable file. However, iOS tends to hide these details about the file system, and as far as we know there is no way to formally ask Xcode where exactly it has installed the app on your device.

So, we use the following workaround:

Open the source file AppDelegate.m, and in the function didFinishLaunchingWithOptions, insert the following line:

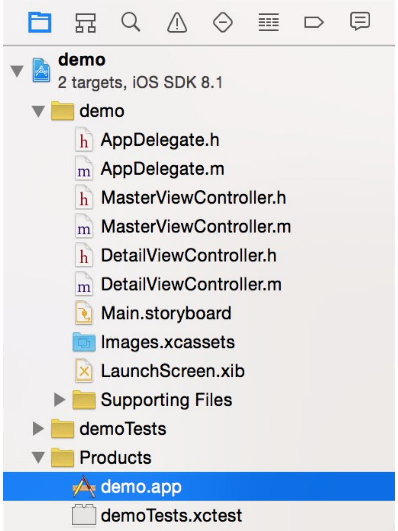
```
NSLog(@"app installation path: %@", [[NSBundle mainBundle] executablePath]);
```

This will ensure that the installation path will be printed in the Xcode console when the app is run.

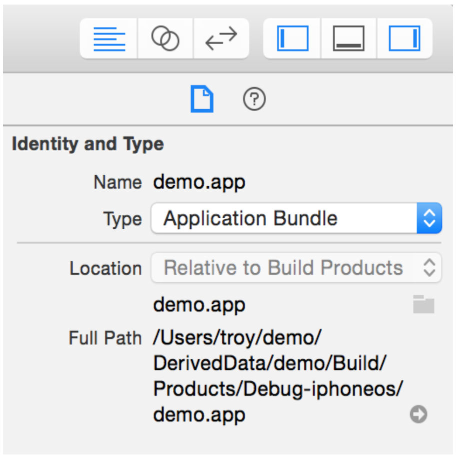
Now click on the big Play button in the top left of the Xcode window. This will build, install, and launch the app on your target device. Once you see that the app has been launched and the application path as been printed to the console, press the Stop button in the top left.

Launching the Debugger

Now it's time to open our sample app in IDA. On the left side of the Xcode window, under the Project Navigator tab, click on demo.app under the Products folder:



```
▼ demo
  2 targets, iOS SDK 8.1
  ▼ demo
      AppDelegate.h
      AppDelegate.m
      MasterViewController.h
      MasterViewController.m
      DetailViewController.h
      DetailViewController.m
      Main.storyboard
      Images.xcassets
      LaunchScreen.xib
    ▶ Supporting Files
  ▶ demoTests
  ▼ Products
      demo.app
      demoTests.xctest
```



Identity and Type

Name: demo.app

Type: Application Bundle

Location: Relative to Build Products

demo.app

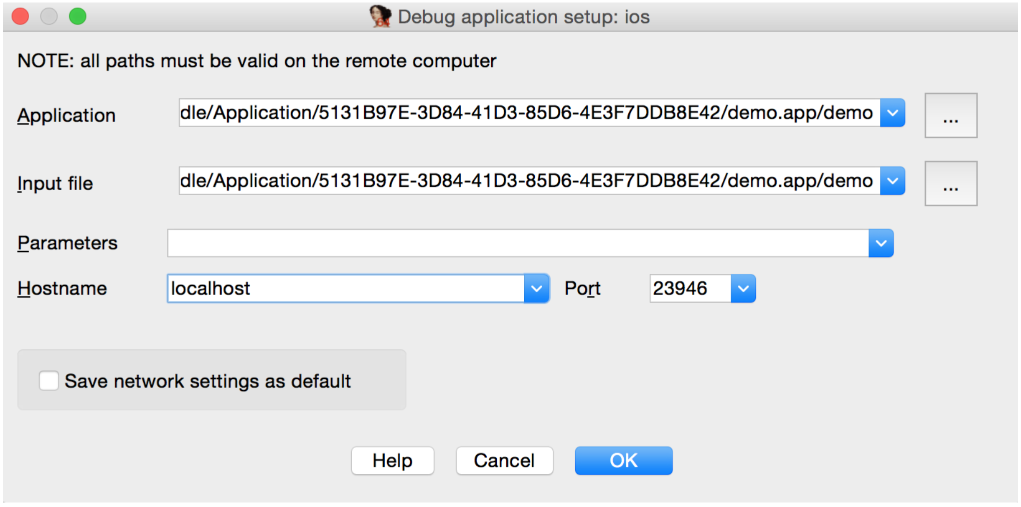
Full Path: /Users/troy/demo/DerivedData/demo/Build/Products/Debug-iphoneos/demo.app

Then, on the right side of the Xcode window under the Utilities tab, you can find the path to newly built app bundle. Use this path to open the app's executable file in IDA:

```
troy@mac:~$ idaq64 /Users/troy/demo/DerivedData/demo/Build/Products/Debug-iphoneos/demo.app/demo
```

Once IDA has finished loading the file, select 'Remote iOS Debugger' from the combo box at the top of the screen and set a breakpoint at main.

Now open menu Debugger>Process options... and for the 'Application' and 'Input File' fields, use the path that was printed to the console when you ran the app in the 'Installation' section above:



Debug application setup: ios

NOTE: all paths must be valid on the remote computer

Application: dle/Application/5131B97E-3D84-41D3-85D6-4E3F7DDB8E42/demo.app/demo

Input file: dle/Application/5131B97E-3D84-41D3-85D6-4E3F7DDB8E42/demo.app/demo

Parameters:

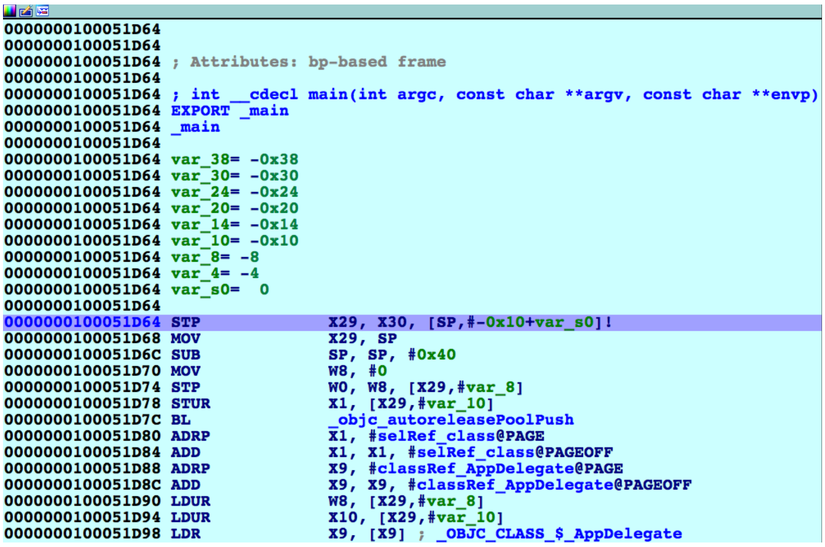
Hostname: localhost

Port: 23946

□ Save network settings as default

In this situation, IDA will not use the 'Hostname' and 'Port' fields, but it still always requires a non-empty hostname. Just set it to 'localhost'.

Now click Debugger>Start process, and wait for the breakpoint at main to be hit:



```
0000000100051D64
0000000100051D64
0000000100051D64 ; Attributes: bp-based frame
0000000100051D64
0000000100051D64 ; int __cdecl main(int argc, const char **argv, const char **envp)
0000000100051D64 EXPORT _main
0000000100051D64 _main
0000000100051D64
0000000100051D64 var_38= -0x38
0000000100051D64 var_30= -0x30
0000000100051D64 var_24= -0x24
0000000100051D64 var_20= -0x20
0000000100051D64 var_14= -0x14
0000000100051D64 var_10= -0x10
0000000100051D64 var_8= -8
0000000100051D64 var_4= -4
0000000100051D64 var_s0=  0
0000000100051D64
0000000100051D64 STP                X29, X30, [SP,#-0x10+var_s0]!
0000000100051D68 MOV                X29, SP
0000000100051D6C SUB                SP, SP, #0x40
0000000100051D70 MOV                W8, #0
0000000100051D74 STP                W0, W8, [X29,#var_8]
0000000100051D78 STUR               X1, [X29,#var_10]
0000000100051D7C BL                 _objc_autoreleasePoolPush
0000000100051D80 ADRP               X1, #selRef_class@PAGE
0000000100051D84 ADD                X1, X1, #selRef_class@PAGEOFF
0000000100051D88 ADRP               X9, #classRef_AppDelegate@PAGE
0000000100051D8C ADD                X9, X9, #classRef_AppDelegate@PAGEOFF
0000000100051D90 LDUR               W8, [X29,#var_8]
0000000100051D94 LDUR               X10, [X29,#var_10]
0000000100051D98 LDR                X9, [X9] ; _OBJC_CLASS_$_AppDelegate
```

And that's it! You can now explore process memory, inspect registers, single step, etc. Happy debugging!

ios_deploy

We recommend Mac users to download 'ios_deploy' from the hex-rays downloads page:

https://www.hex-rays.com/products/ida/support/ida/ios_deploy.zip

This helper utility has effectively replaced Xcode in our development cycle, and can perform all the everyday tasks that can be difficult on iOS - like signing and installing applications, extracting debugging symbol files from dyld_shared_cache, and retrieving the installation paths of target executables.

See ios_deploy -h to get a quick look at what it can do.

The goal of this part of the tutorial is to use ios_deploy to install a prebuilt sample application and debug it in IDA – with limited dependence on Xcode.

Connection

Before we get started lets do a sanity check to make sure IDA will be able to recognize and connect to your device. Connect a device to your Mac via USB (currently ios_deploy cannot work over Wi-Fi), and run the following command:

```
troy@mac:~$ ios_deploy listen

Device connected:
- name:    iPhone 6
- model:   iPhone 6s
- ios ver:  10.0
- build:    14A5341a
- arch:     arm64
- id:       9b72866777b672d81bcf080964926f3b9864a9b2
^C
troy@mac:~$
```

ios_deploy should print a message every time a device is connected/disconnected. If you see a message like you see above, then so far so good.

Installing the debugserver

The next step is to ensure that the debugserver has been installed on your device and IDA will be able to launch it. You can check this using the 'diagnostics' phase:

```
troy@mac:~$ ios_deploy diagnostics
Device: iPhone 6 (iPhone 6s, iOS 10.0)
Warning: could not launch the debugserver (The service is invalid.)
Trying to re-mount the developer disk image...
Error: could not open signature file
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/10.0
(14A5341a)/DeveloperDiskImage.dmg.signature
Phase 'diagnostics' failed, exiting
troy@mac:~$
```

If the debugserver is not present on your device, you will get a 'service is invalid' message, and the diagnostics phase will try install the DeveloperDiskImage.dmg automatically and try again.

If diagnostics failed to find a matching DeveloperDiskImage.dmg for your device (like it did above), you can find one manually. They are usually located in:

/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/

Then use 'mount -d <path>' to install it:

```
troy@mac:~$ ios_deploy mount -d \
> /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/10.0\ \
(14A5339a\)/DeveloperDiskImage.dmg
troy@mac:~$ ios_deploy diagnostics
Device: iPhone 6 (iPhone 6s, iOS 10.0)
Diagnostics completed. No issues to report
troy@mac:~$
```

If the debugserver could be launched, you should see 'Diagnostics completed. No issues to report'.

Symbols

The next step is to extract debug symbols from dyld_shared_cache. IDA relies heavily on these symbol files in order to achieve fast and detailed debugging (as does Xcode – it usually stores symbols in ~/Library/Developer/Xcode/iOS\ DeviceSupport when you first connect your device).

To extract the symbol files to your host, run the “symbols” phase:

```
troy@troy:~$ ios_deploy symbols
Copying dyld...
Downloading 0.59 MB of 0.59 MB
/tmp/dyld: done
Copying dyld_shared_cache_arm64...
Downloading 767.19 MB of 767.19 MB
/tmp/dyld_shared_cache_arm64: done
Extracting symbol file: 957/1197
```

When this phase completes it will store the symbols at ~/Symbols. We can now tell IDA where to find them by setting SYMBOL_PATH in idaq.app/Contents/MacOS/cfg/dbg_ios.cfg:

```
 SYMBOL_PATH = "~/Symbols";
```

Installation

Now it's time to set up an example target application. We have provided a prebuilt example iOS binary in 'ios_demo' at https://www.hex-rays.com/products/ida/support/ida/ios_demo.zip, along with the source code. You must codesign this application in order to install and debug it on your device, which means you must be part of the iPhone Developer program.

If the application is not properly codesigned with your developer certificate, the debugserver will refuse to debug it, reporting a 'Failed to get task for process' error message.

See 'ios_deploy codesign -h' for more on how this process works.

Once you have verified that you have an iPhone Developer certificate and you have downloaded the example app, cd to ios_demo/ and run the following command:

```
troy@mac:~/ios_demo$ ios_deploy appify -e demo
```

This will create an app bundle at ./demo.app, which can then be installed on your device with:

```
troy@mac:~/ios_demo$ ios_deploy install -b demo.app/
```

You can then print the installation path via:

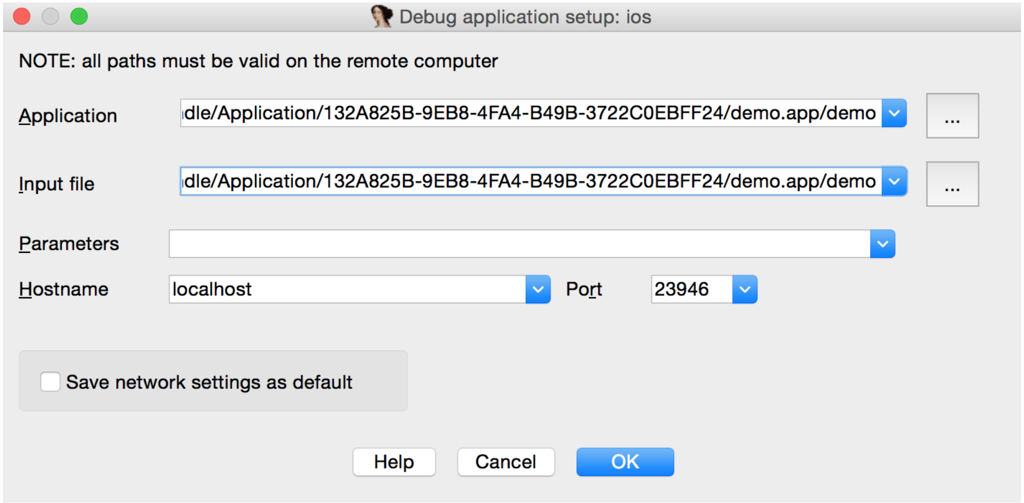
```
troy@troy:~/ios_demo$ ios_deploy paths -b demo -s
/var/containers/Bundle/Application/132A825B-9EB8-4FA4-B49B-3722C0EBFF24/demo.app/demo
```

Launching the Debugger

Finally, it's time to launch IDA and run the debugger. First open the example app in ida:

```
troy@troy:~/ios_demo$ idaq demo.app/demo
```

Once IDA has finished loading the file, Select 'Remote iOS Debugger' from the combo box at the top of the window and set a breakpoint at main. Open menu Debugger>Process options... and for both the 'Application' and 'Input file' fields, insert the path you retrieved by running 'ios_deploy paths' above:



Debug application setup: ios

NOTE: all paths must be valid on the remote computer

Application: dle/Application/132A825B-9EB8-4FA4-B49B-3722C0EBFF24/demo.app/demo

Input file: dle/Application/132A825B-9EB8-4FA4-B49B-3722C0EBFF24/demo.app/demo

Parameters:

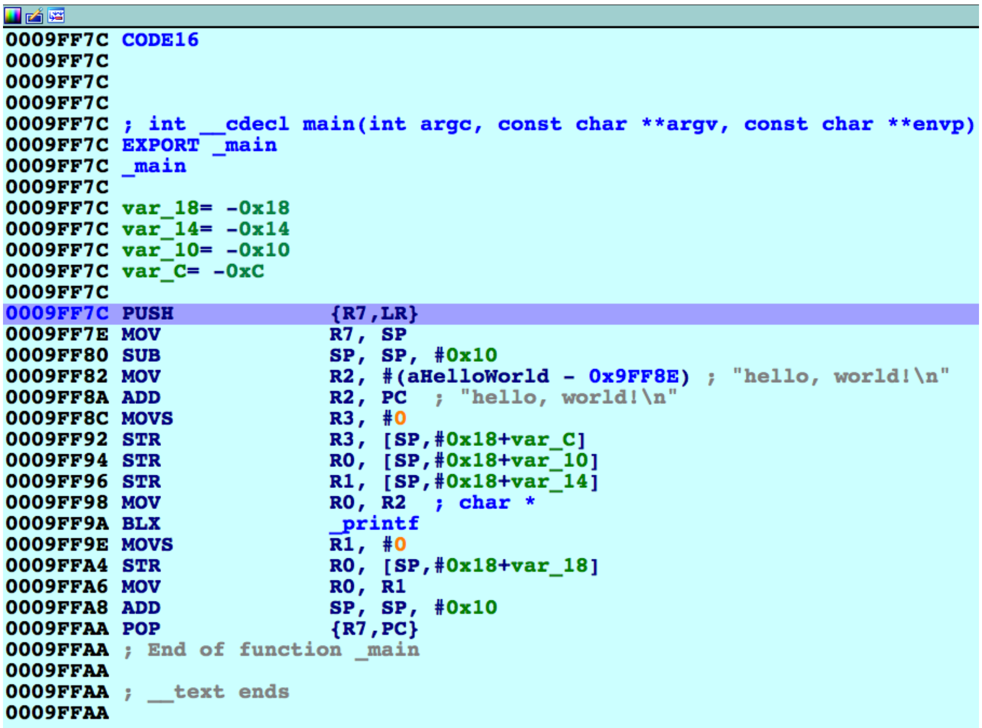
Hostname: localhost

Port: 23946

☐ Save network settings as default

In this situation, IDA will not use the 'Hostname' and 'Port' fields, but it still always requires a non-empty hostname. Just set it to 'localhost'.

Now click Debugger>Start process, and wait for the breakpoint at main to be hit:



```
0009FF7C CODE16
0009FF7C
0009FF7C
0009FF7C
0009FF7C ; int __cdecl main(int argc, const char **argv, const char **envp)
0009FF7C EXPORT _main
0009FF7C _main
0009FF7C
0009FF7C var_18= -0x18
0009FF7C var_14= -0x14
0009FF7C var_10= -0x10
0009FF7C var_C= -0xC
0009FF7C
0009FF7C PUSH                {R7,LR}
0009FF7E MOV                 R7, SP
0009FF80 SUB                 SP, SP, #0x10
0009FF82 MOV                 R2, #(aHelloWorld - 0x9FF8E) ; "hello, world!\n"
0009FF8A ADD                 R2, PC   ; "hello, world!\n"
0009FF8C MOVS                R3, #0
0009FF92 STR                 R3, [SP,#0x18+var_C]
0009FF94 STR                 R0, [SP,#0x18+var_10]
0009FF96 STR                 R1, [SP,#0x18+var_14]
0009FF98 MOV                 R0, R2   ; char *
0009FF9A BLX                 _printf
0009FF9E MOVS                R1, #0
0009FFA4 STR                 R0, [SP,#0x18+var_18]
0009FFA6 MOV                 R0, R1
0009FFA8 ADD                 SP, SP, #0x10
0009FFAA POP                 {R7,PC}
0009FFAA ; End of function _main
0009FFAA
0009FFAA ; __text ends
0009FFAA
```

And that's it! You can now step, explore process memory, inspect registers, etc. Happy debugging!

Troubleshooting

Process Launch

If IDA fails to launch the target process for whatever reason, it will usually print an error message to the console window. Here are some common error messages:

- **Elocked -** this means the debugserver failed to launch the process because the device's screen is locked. Simply unlock the screen and this error should go away.
- **The service is invalid** - this usually means IDA tried to launch a debugging utility that is not installed on the device. Please ensure that the DeveloperDiskImage.dmg has been mounted on your device (either via Xcode or 'ios_deploy mount').
- **Efailed to get the task for process** - this likely means that the debugserver does not have permission to debug the target app. Please ensure that the target app has been properly codesigned for debugging ('ios_deploy codesign' can do this for you).

  SA: http://iphonedevwiki.net/index.php/Debugserver and AUTOLAUNCH in dbg_ios.cfg

IDA_DEBUG_DEBUGGER

This flag will make IDA print very verbose logging messages to the console while the debugger is running. Enable it by launching IDA with: '/path/to/idaq -z10000'

If this option is enabled when using the Remote iOS debugger, IDA will log all of the packet communication between IDA and the debugserver. Look for lines prefixed with:

> … (data sent to debugserver)

and

< … (data received from debugserver)

Often times these packets will contain messages or error codes that might describe the problem.

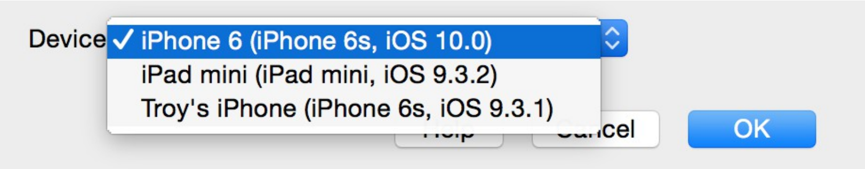
Syslog

You can also use the SYSLOG_FLAGS option in dbg_ios.cfg to instruct the debugserver to print extra debug messages to the iOS system log.

You can then use 'ios_deploy syslog' while IDA is running to fetch the system log.

Working with Multiple Devices

Be careful when working with multiple iOS devices connected to your host simultaneously. In this situation you can select the target device via menu Debugger>Debugger options>Set specific options:



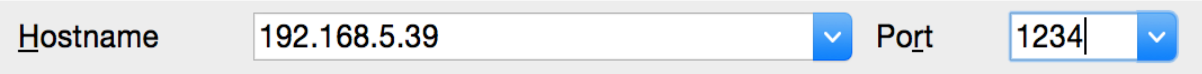
Device ✓ iPhone 6 (iPhone 6s, iOS 10.0)
iPad mini (iPad mini, iOS 9.3.2)
Troy's iPhone (iPhone 6s, iOS 9.3.1)

If you have multiple devices connected and you don't specify a target device, IDA will simply use the first device it finds. The device used is not guaranteed to be deterministic, so its a good idea to explicitly tell IDA which device to use.

Non-Mac users

On non-mac platforms, IDA has no way to launch the debugserver automatically like it does on Mac. Thus, you must be able to ssh to your device and launch the debugserver yourself, specifying a port to listen on. Then specify this port and the hostname of your device in the Process Options dialog:

```
testers-iPad:~ root# ./debugserver *:1234
debugserver-64 for armv6 Copyright (c) 2007-2009 Apple, Inc.  All Rights Reserved.
Listening to port 1234...
```



Hostname 192.168.5.39 Port 1234

See http://iphonedevwiki.net/index.php/Debugserver for how to get the most out of the debugserver.

Also see http://iphonedevwiki.net/index.php/SSH_Over_USB for an overview of how you can ssh to your device over USB.

Symbols

IDA relies heavily on symbol files extracted from dyld_shared_cache for fast and detailed debugging. It is definitely possible to debug without symbol files, but IDA will be much slower and will be missing many symbol names.

If you manage to extract symbol files from your device (either using a third-party tool or Xcode if you have access to a Mac), you can tell IDA where to find these files via the SYMBOL_PATH variable in dbg_ios.cfg.